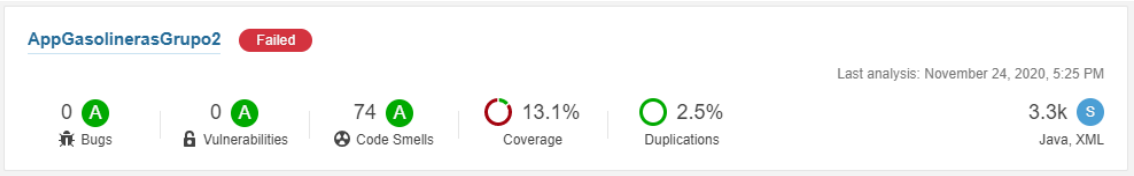
INFORME DE CALIDAD (SPRINT 3)

AUTOR: RUBEN CALLEJA Y DANIEL SÁNCHEZ

Análisis 24 noviembre 2020

Captura



Incidencias

El análisis no paso los criterios de calidad de la organización debido a que la deuda técnica, supera las 4 horas establecidas. Nos encontramos con una deuda tecina de 1 día y 4 horas generada por los 74 *code smells* presentes en el código. Dichos *code smells*, se encuentran bastante distribuidos, pero podemos observar, que el mayor número se encuentra en las clases *MainActivity* y *PresenterVehiculos*, ya que ambas contienen 18, ya que la siguiente que más tiene es *MisVehiculosActivity*, con 8.

Por otra parte, si nos fijamos en la severidad de dichos errores, observamos que solo 8 son críticos y 15 mayores, mientras que existen un gran número de errores menores y de información, situándose estos en 24 y 27 respectivamente.

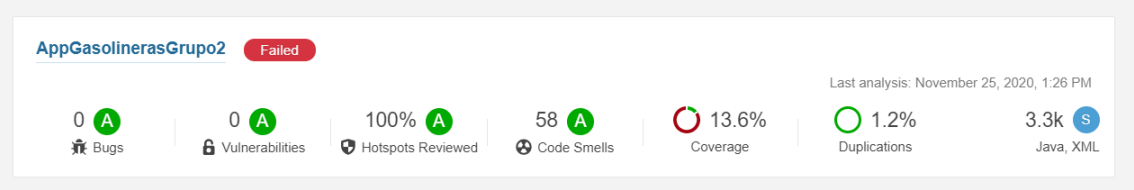
Plan de acción

1. Eliminar duplicación de código existente en *MainActivity*, ya que genera una deuda de 30 minutos y es código repetido que no es nada aconsejable tener. Para ello se propone crear un método que aporte esta funcionalidad y que se invoque en los dos lugares donde es necesario.
2. Eliminar las impresiones por pantalla con *System.out.println()* sustituyéndolas por *Log.d()* que es más apropiado.
3. Eliminar los *imports* innecesarios de todas las clases, ya que se elimina de manera muy rápida y conseguimos así eliminar la deuda técnica que generan.
4. Eliminar también ciertas líneas que están comentadas y no son necesarias (“*This block of commented-out lines of code should be removed”),* para clarificar el código y reducir la deuda técnica.

Rubén Calleja Reigadas

Análisis 25 noviembre 2020

Captura



Incidencias

Podemos observar que el código no cumple los estándares de calidad propuestos por la organización. Esto se debe a que, aunque todos los criterios están calificados como A, existe una deuda técnica superior a las 4 horas establecidas. Si se observa con más detalle se puede analizar el porqué de esta deuda, que en este caso es que existen 58 *code smells*, de los cuales 10 son de categoría crítica y 7 son de categoría grave. Estos *code smells* se encuentran principalmente en *MainActivity*, *PresenterVehiculos* y *MisVehiculosActivity*, aunque hay alguno en otras localizaciones que crean bastante deuda, como por ejemplo entre las clases *DieselFiltro* y *Gasolina95Filtro*, que son dos clases completamente idénticas y genera una duplicación de código.

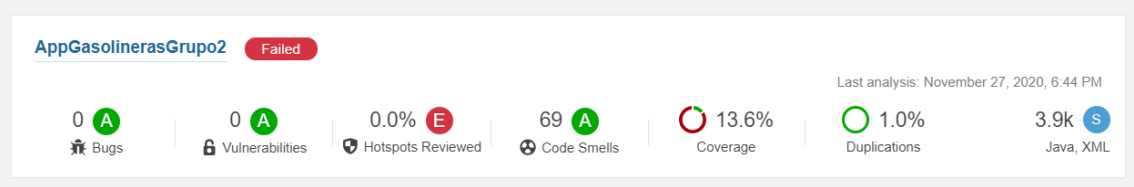
Plan de acción

1. Centrarse en el *PresenterVehiculos* para eliminar los fallos críticos relativos a la Entrada/Salida de archivos, lo cual también cubrirá algún fallo mayor. Para ello se cambiarán los try/catch y se cambiarán las llamadas a ciertos métodos de borrado.
2. Cambiar el string “Error” que se repite varias veces por una constante.
3. Añadir un comentario y una excepción a un método vacío en *FormActivity*.
4. Cambiar ciertas llamadas estáticas en los permisos para llamar a clases padre en vez de las hijas en el *MainActivity*.
5. Cambiar en *FilterActivity* los getters innecesarios por constantes.

Daniel Sánchez Díez

Análisis 28 noviembre 2020

Captura



Incidencias

El código no cumple las condiciones necesarias para pasar los criterios de calidad, y hay varias razones para ello. Para empezar, se puede ver que el criterio de Security tiene una calificación de E, y si observamos más en detalle se observa que existen 2 Security Hotspots. Estos se deben a que existen dos instancias de un catch cuyo cuerpo tiene la línea: e.printStackTrace. Esto es un problema porque proporciona a desarrolladores externos información detallada de como exactamente falla la aplicación, lo cual puede ser explotado. La solución es simplemente cambiarlo por un Log.D corriente.

Asimismo, se puede observar que existe una deuda técnica de un día y cuatro horas, que excede con creces el límite establecido. Dicha deuda toma forma en los 69 code smells que existen, de los cuales 8 son críticos y 6 graves. Estos smells se encuentran mayoritariamente en *MainActivity*, *PresenterVehiculos* y *MisVehiculosActivity* pero también hay fallos menos graves que añaden mucha deuda en otros sitios.

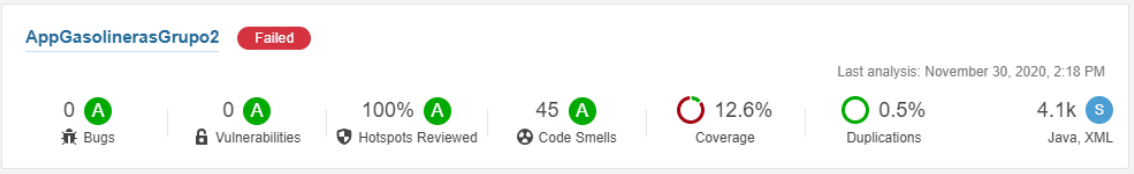
Plan de acción

1. Eliminar los security hotspots de MainActivity y MisVehiculosActivity.
2. En MainActivity y PresenterVehiculos cambiar todos los try/catch por try/catch with resources (cambiando algún detalle de la estructura de los métodos donde se encuentran para que sigan funcionando).
3. Cambiar la estructura de las clases DieselFiltro y Gasolina95Filtro para que en vez de tener código duplicado utilicen una clase común.
4. En MainActivity:
    a. Cambiar varios strings duplicados por constantes.
    b. Refactorización del método onPostExecute para reducir su complejidad hasta unos valores aceptables.
    c. Eliminar variable no usada eliminada.
5. En FilterActivity:
    a. Refactorización del método onCheckBoxClicked para reducir su complejidad hasta unos valores aceptables.
    b. Cambiar 4 constantes a final, ya que no se modifican nunca. Están marcados como minor, pero cada una de ellas suma 20 minutos.
6. En la clase de utilidad Distancia añadir un constructor privado que lance una excepción de IllegalState para ocultar el constructor implícito.

Daniel Sánchez Díez

Análisis 30 noviembre 2020

Captura



Incidencias

El análisis no paso los criterios de calidad de la organización debido a que la deuda técnica, supera las 4 horas establecidas. Nos encontramos con una deuda tecina de 1 día, generada por los 45 *code smells* presentes en el código. Dichos *code smells*, se encuentran centralizados en su mayoría en *MainActivity* y *MisVehiculosActivity*, teniendo estas 17 y 8 respectivamente.

Aunque gran parte de ellos son por métodos *deprecated* que no se pueden sustituir.

Por otra parte, si nos fijamos en la severidad de dichos errores, observamos que solo 2 son mayores, mientras que existen un gran número de errores menores y de información, situándose estos en 15 y 28 respectivamente.

Por otra parte, tenemos varios *code smells* repetidos o muy similares en las clases de Pop-up que se solucionarían de la misma manera.

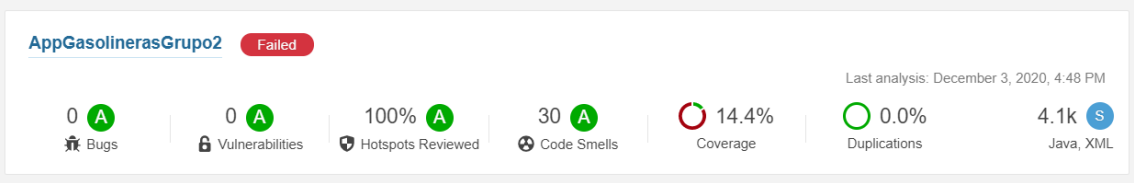
Plan de acción

1. Eliminar los bloques de código repetidos en *PopUpConflicto* y *PopUpPrimerVehiculo*.
2. Eliminar los *imports* no usados.
3. Eliminar comentarios de TODO e innecesarios.
4. Mover los métodos que están en *MainActivity* al método *cargaDatosGasolinerasTask*.
5. Renombrar las variables que no cumplen con el estándar.
6. Renombrada clase presenterVehiculos, para cumplir con el estándar.

Rubén Calleja Reigadas

Análisis 3 de diciembre 2020

Captura



Incidencias

El análisis nos indica que el código no pasa los criterios de calidad, ya que la deuda técnica que sitúa en 4 horas y 42 minutos. Esto surge por culpa de los 30 *code smells*, de los cuales 5 eran de severidad *blocker*, porque al comentar los tests de varias clases de prueba, que funcionan en local, pero no en Travis, han quedado vacías.

Aparte de esto hay 10 *code smells* menores y 15 de *info*, siendo los últimos inevitables, ya que saltan porque se usan métodos de una versión de Android obsoleta (añaden aproximadamente una hora y media tanto en *MainActivity* como en *MisVehiculosActivity*).

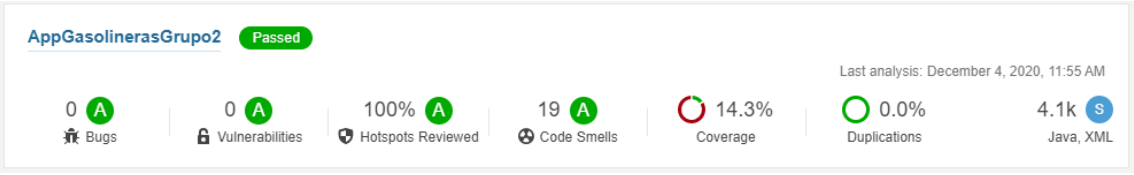
Plan de acción

1. Añadir en las clases de prueba vacías unos métodos que siempre pasen, para que al menos haya algo en ellas.
2. Eliminar los *imports* no usados.
3. En la clase Gasolinera cambiar una estructura *if/else* por un *return boolean*.
4. En *FormActivity* cambiar un atributo de público a privado, ya que no se usa fuera de esa clase.
5. Eliminar un atributo no utilizado en *MainActivity*.
6. Previamente se había reducido en otra rama la deuda gracias a la eliminación del uso de métodos obsoletos para establecer tamaños de ventana, lo que quitó aproximadamente 2 horas de deuda.

Daniel Sánchez Díez

Análisis 4 de diciembre 2020

Captura



Incidencias

El análisis nos indica, que el proyecto cumple con todos los criterios de calidad establecido por la organización. Siendo los únicos *code smells* presentes en el código culpa del código inicialmente proporcionado y duplicado en otra clase por necesidad. Los *code smells* en cuestión son los referidos a la clase *AsyncTask* y sus métodos como el *execute*. Ademas, hay otros tres *code smells* presentes que tienen que ver con que los *adapter* están públicos, pero no se pueden poner privados y acceder a ellos con métodos privados *getter* porque no podemos actualizar el *gradle*. Por último, hay otro *code smell* bloqueante debido a que en una clase no hay ningún

test, pero esto es porque están comentados debido a un problema el cual acordamos solucionar comentando todos los test de interfaz de usuario con varios profesores del proyecto.

Plan de acción

Eliminar el error bloqueante, poniendo un *assertTrue* en el test correspondiente, en este caso, en la clase *ManejoFiltrosITest*.

Realmente no seria necesario, porque la clase tiene tests, pero al estar comentados salta el *issue.*

Rubén Calleja Reigadas

Comentario general:

La mayoría de los momentos en los que el proyecto no cumplía con los criterios de calidad de la organización, es debido a los *code smellls* generados por método *deprecated* como los comentados anteriormente, es decir, los métodos *excute* y la clase *AsyncTask*, además de los métodos *GetMetrics* que ya en los últimos informes, no aparecen porque les solucionamos de una forma que nos comentó Juan. Los *code smells* mencionados anteriormente suponían una deuda técnica de aproximadamente cinco horas y media.